

Polyceptron: A Polyhedral Learning Algorithm

Naresh Manwani

Electrical Engineering Department
Indian Institute of Science
Bangalore, India 560012
Email: naresh@ee.iisc.ernet.in

P. S. Sastry

Electrical Engineering Department
Indian Institute of Science
Bangalore, India 560012
Email: sastry@ee.iisc.ernet.in

Abstract—In this paper we propose a new algorithm for learning polyhedral classifiers which we call as *Polyceptron*. It is a Perception like algorithm which updates the parameters only when the current classifier misclassifies any training data. We give both batch and online version of Polyceptron algorithm. Finally we give experimental results to show the effectiveness of our approach.

I. INTRODUCTION

Learning polyhedral classifier is an interesting problem in machine learning. The need of learning a polyhedral classifier for a binary classification problem arise when the positive examples are all concentrated in a single convex region with the negative examples being all around that region. Even when the positive examples are concentrated in a set which is approximately convex, learning polyhedral classifier would be a useful strategy. For a binary classification problem, a given set of examples is polyhedrally separable if there is a convex polyhedral set that contains all positive examples and no negative example [1].

One can learn a classifier for polyhedrally separable data using support vector data description (SVDD) method [2], which is a variant of the well known Support Vector Machine (SVM) method [3]. SVDD method does this task by fitting a minimum enclosing hypersphere in the feature space to include most of the positive examples inside the hypersphere [2] and all the negative examples are considered as outliers. In such techniques, the nonlinearity in the data is captured simply by choosing an appropriate kernel function. Although the SVM methods often give good classifiers, with a non-linear kernel function, the final classifier may not provide good geometric insight on the class boundaries in the original feature space. Learning each of the hyperplanes that make up the polyhedral set is often useful to understand geometry of the class region and the local behavior of the classifier in different regions of the feature space.

Decision tree is another approach which can be used to learn polyhedral classifiers. When all positive examples belong to a single polyhedral set, the ideal decision tree learnt would be such that every non-leaf node has one of the children as a leaf (representing negative class) and there is only one path leading to a leaf for the positive class. Such a decision tree is called a decision list and it represents the polyhedral classifier exactly.

Polyhedral classifier can be learnt using decision tree in two ways. The first one is to use one of the top down greedy method which is followed in many decision tree algorithms [4], [5]. In the top down approaches, the impurity based heuristics are used to learn optimal hyperplanes at each node. But such general decision tree algorithm fails to learn a single polyhedral set well. That is, the learnt decision tree may be a general tree and not a decision list.

In the second approach the tree structure is fixed and optimal parameters are learnt for the tree.

For the polyhedrally separable data, the decision tree learning can be reformulated as learning a decision list of fixed structure. The structure is fixed by assuming that the number of hyperplanes that make the required polyhedral set are known beforehand. One possible choice for learning such a decision list is to formulate a constrained optimization problem [6], [7], [8]. The objective there is to minimize the classification errors subject to the separability conditions. It is worth noting that these optimization problems are non-convex even though we are learning a convex set. Here all the positive examples have to satisfy each of a given set of linear inequalities. Thus the constraint on each of the positive examples is logical ‘and’ of the linear constraints and hence form a convex set. However, each of the negative examples fail to satisfy one (or more) of these inequalities and a priori it is not known which inequality each negative example fails to satisfy. Thus constraint on each of the negative examples is logical ‘or’ of the linear constraints and hence form a nonconvex set. In a logical ‘or’ constraint for a negative example, identifying which of the linear constraints is violated, is called the credit assignment problem in the literature and it makes learning polyhedral sets a difficult task [1].

In [6], this problem is solved by first enumerating all possibilities for misclassified negative examples (e.g., which of the hyperplanes caused each negative example to get misclassified and for each negative example there could be many such hyperplanes) and then solving a linear program for each possibility to find descent direction. This approach becomes computationally very expensive.

If, for every point falling outside the polyhedral set, it is known beforehand which of the linear inequalities it will satisfy, then the problem becomes much easier. In that case, the problem becomes one of solving K linear classification problems independently. But this assumption is very unrealistic.

[8] relaxes this assumption a little and assumes that for each sub-classification problem corresponding to every hyperplane, a small subset of negative examples is known and propose a cyclic optimization algorithm (optimizing one classifier out of K at a time). Still, their assumption of knowing subset of negative examples corresponding to each hyperplane is not realistic in many practical applications.

Recently, a probabilistic discriminative model has been proposed in [9] using logistic function to learn a polyhedral classifier. It is an unconstrained framework and a simple expectation maximization algorithm is followed to learn the parameters. But still the approach proposed in [9] is a batch algorithm and there is no incremental variant of this algorithm.

In this paper we propose a Perceptron like algorithm to learn polyhedral classifier which we call *Polyceptron*. We present the Polyceptron criterion which is minimized by Polyceptron algorithm.

The Polyceptron criterion is designed in the same way as Perceptron criterion. Perceptron algorithm learns linear classifier by minimizing the Perceptron criterion in which only misclassified points contribute to the error term. In the Polyceptron criterion also, only misclassified points contribute to the error term. In other words, it assigns zero error for correctly classified points. Since we have modified the Perceptron algorithm to learn **polyhedral** classifiers, we call it **Polyceptron**.

In this paper, we propose both batch and online version of the Polyceptron algorithm. The batch Polyceptron is an alternating minimization algorithm to minimize the Polyceptron criterion. We also present the online Polyceptron algorithm which treats one sample at a time. In the online Polyceptron the hyperplane parameters are updated only when the current example is misclassified.

The rest of the paper is organized in the following way. In section II we discuss polyhedral separability and define polyhedral classifier. We describe the Polyceptron algorithm in section III. Experiment results are discussed in section IV. Finally we conclude the paper with some discussions in section V.

II. POLYHEDRAL CLASSIFIER

Let $D = \{(\mathbf{x}_n, t_n) : \mathbf{x}_n \in \mathbb{R}^d ; t_n \in \{-1, 1\}, n = 1 \dots N\}$ be the training dataset. Let \mathcal{A} be the set of points for which $t_n = 1$. Also let \mathcal{B} be the set of points for which $t_n = -1$. First we restate the polyhedral separability defined in [1], [6].

A. Polyhedral Separability

Two sets \mathcal{A} and \mathcal{B} in \mathbb{R}^d are K -polyhedral separable if there exists a set of K hyperplanes having parameters (\mathbf{w}_k, b_k) , $k = 1 \dots K$ with $\mathbf{w}_k \in \mathbb{R}^d$, $b_k \in \mathbb{R}$, $k = 1 \dots K$ such that

- 1) $\mathbf{w}_k^T \mathbf{x} + b_k \geq 0$, $\forall \mathbf{x} \in \mathcal{A}$, $k = 1 \dots K$
- 2) $\mathbf{w}_k^T \mathbf{x} + b_k < 0$, $\forall \mathbf{x} \in \mathcal{B}$, for at least one $k \in \{1, \dots, K\}$

This means that two sets \mathcal{A} and \mathcal{B} are K -polyhedral separable if \mathcal{A} is contained in a convex polyhedral set which is formed

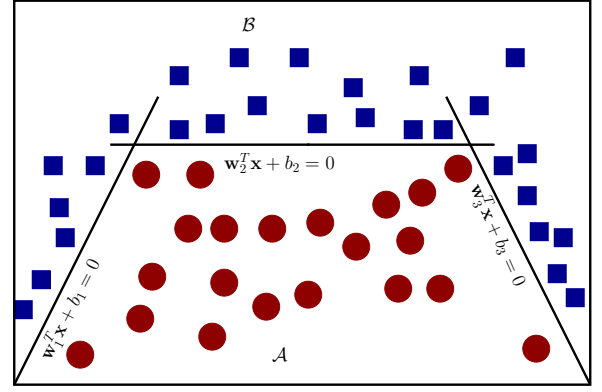


Fig. 1. An example of polyhedrally separable sets \mathcal{A} and \mathcal{B}

by intersection of K halfspaces and the points of set \mathcal{B} are outside this polyhedral set. Figure.1 shows an example of two sets \mathcal{A} and \mathcal{B} that are 3-polyhedrally separable.

B. Polyhedral Classifier

Let $\tilde{\mathbf{w}}_k = [\mathbf{w}_k \ b_k]^T \in \mathbb{R}^{d+1}$ and let $\tilde{\mathbf{x}}_n = [\mathbf{x}_n \ 1]^T \in \mathbb{R}^{d+1}$. We now express the earlier inequalities as $\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}} > 0$ and so on. Using the definition of polyhedral separability discussed earlier, let us define a function $h(\mathbf{x}, \Theta)$ as below

$$h(\mathbf{x}, \Theta) = \min_{k \in \{1, \dots, K\}} (\mathbf{w}_k^T \mathbf{x} + b_k)$$

where $\Theta = \{\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K\}$ is the set of parameters of the K hyperplanes. Clearly, if $h(\mathbf{x}, \Theta) \geq 0$, then the condition $\mathbf{w}_k^T \mathbf{x} + b_k \geq 0$, $\forall k = 1 \dots K$ is satisfied and the point \mathbf{x} will be assigned to set \mathcal{A} . Similarly if $h(\mathbf{x}, \Theta) < 0$, there exists at least one k for which $\mathbf{w}_k^T \mathbf{x} + b_k < 0$ and the point \mathbf{x} will be assigned to set \mathcal{B} . Let us assume that we know K (number of hyperplanes forming the polyhedral set). Then the polyhedral classifier will become $f(\mathbf{x}, \Theta) = \text{sign}(h(\mathbf{x}, \Theta))$.

Now the problem is to learn Θ , the set of parameters of all the hyperplanes, given the training data.

III. POLYCEPTRON

Here we propose a Perceptron like algorithm for learning polyhedral classifier which we call *Polyceptron*. The goal of Polyceptron is to find the parameter set $\Theta = \{\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K\}$ of K hyperplanes such that point $\mathbf{x}_n \in \mathcal{A}$ will have $h(\mathbf{x}_n, \Theta) = \min_{k \in \{1, \dots, K\}} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n) > 0$, whereas point $\mathbf{x}_n \in \mathcal{B}$ will have $h(\mathbf{x}_n, \Theta) = \min_{k \in \{1, \dots, K\}} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n) < 0$. Since $t_n \in \{-1, 1\}$ is the class label for \mathbf{x}_n , we want that each point \mathbf{x}_n should satisfy $t_n h(\mathbf{x}_n, \Theta) > 0$.

Polyceptron algorithm finds polyhedral classifier by minimizing the *Polyceptron criterion* which is defined as follows.

$$E_P(\Theta) := - \sum_{n=1}^n t_n h(\mathbf{x}_n, \Theta) I_{\{t_n h(\mathbf{x}_n, \Theta) < 0\}}$$

Where $I_{\{t_n h(\mathbf{x}_n, \Theta) < 0\}}$ is an indicator function which takes value '1' if $t_n h(\mathbf{x}_n, \Theta) < 0$ and '0' otherwise. Polyceptron criterion assigns zero error for a correctly classified point. On

the other hand, if a point \mathbf{x}_n is misclassified, the Polyceptron criterion tries to minimize the quantity $-t_n h(\mathbf{x}_n, \Theta)$.

A. Batch Polyceptron

Batch Polyceptron minimizes the Polyceptron criterion considering all the data points at a time to find the parameters of the polyhedral classifier. Batch Polyceptron works in the following way.

Given parameters of K hyperplanes, $\tilde{\mathbf{w}}_1 \dots \tilde{\mathbf{w}}_K$, define sets $S_k = \{\mathbf{x}_n | \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k \leq \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j, \forall j \neq k\}$ where we break ties by putting \mathbf{x}_n in the set S_k with least k if $\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k \leq \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j, \forall j \neq k$ is satisfied by more than one $k \in \{1, \dots, K\}$. The sets S_k are disjoint. We can now write $E_P(\Theta)$ as

$$E_P(\Theta) = - \sum_{k=1}^K \sum_{\mathbf{x}_n \in S_k} t_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k I_{\{t_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k < 0\}} \quad (1)$$

For a fixed k , $-\sum_{\mathbf{x}_n \in S_k} t_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k I_{\{t_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k < 0\}}$ is same as the Perceptron criterion function and we can find $\tilde{\mathbf{w}}_k$ to optimize this by using Perceptron algorithm. However, in $E_P(\Theta)$ defined by (1), the sets S_k themselves are function of the set of parameters $\Theta = \{\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K\}$. Hence we can not directly minimize $E_P(\Theta)$ given by (1) using standard gradient descent.

To minimize the Polyceptron criterion we adopt an alternating minimization scheme in the following way. Let after c^{th} iteration, the parameter set be Θ^c . Keeping Θ^c fixed we calculate the sets $S_k^c = \{\mathbf{x}_n | \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^c \leq \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^c, \forall j \neq k\}$. Now we keep these sets S_k^c fixed. Thus the Polyceptron criterion after c^{th} iteration becomes

$$\begin{aligned} E_P^c(\Theta) &= - \sum_{k=1}^K \sum_{\mathbf{x}_n \in S_k^c} t_n \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n I_{\{t_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k < 0\}} \\ &= \sum_{k=1}^K f_k^c(\tilde{\mathbf{w}}_k) \end{aligned}$$

where $f_k^c(\tilde{\mathbf{w}}_k) = -\sum_{\mathbf{x}_n \in S_k^c} t_n \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n I_{\{t_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k < 0\}}$. Super-script c is used to emphasize that the Polyceptron criterion is evaluated by fixing the sets S_k^c , $k = 1 \dots K$. Thus $E_P^c(\Theta)$ becomes a sum of K functions $f_k^c(\tilde{\mathbf{w}}_k)$ in such a way that $f_k^c(\tilde{\mathbf{w}}_k)$ depends only on $\tilde{\mathbf{w}}_k$ and it does not vary with the other $\tilde{\mathbf{w}}_j$, $\forall j \neq k$.

Now minimizing $E_P^c(\Theta)$ with respect to Θ boils down to minimizing each of $f_k^c(\tilde{\mathbf{w}}_k)$ with respect to $\tilde{\mathbf{w}}_k$. For every $k \in \{1, \dots, K\}$, a new weight vector $\tilde{\mathbf{w}}_k^{c+1}$ is found using gradient descent update as follows.

$$\begin{aligned} \tilde{\mathbf{w}}_k^{c+1} &= \tilde{\mathbf{w}}_k^c - \eta(c) \frac{\partial E_P^c}{\partial \tilde{\mathbf{w}}_k} \\ &= \tilde{\mathbf{w}}_k^c + \eta(c) \sum_{\mathbf{x}_n \in S_k^c} t_n \tilde{\mathbf{x}}_n \end{aligned}$$

where $\eta(\cdot)$ is the step size. Here we have given only one iteration of gradient descent. We may not minimize $f_k^c(\tilde{\mathbf{w}}_k)$ exactly, so we run a few steps of gradient descent to make sure that the newly found weight vector $\tilde{\mathbf{w}}_k^{c+1}$ is such that $f_k^c(\tilde{\mathbf{w}}_k^{c+1}) < f_k^c(\tilde{\mathbf{w}}_k^c)$. Then we calculate S_k^{c+1} and so on.

To summarize, batch Polyceptron is an alternating minimization algorithm to minimize the Polyceptron criterion. This algorithm first finds the sets S_k^c , $k = 1 \dots K$ for iteration c and then for each $k \in \{1, \dots, K\}$ it learns a linear classifier by minimizing $f_k^c(\tilde{\mathbf{w}}_k)$. We keep on repeating these two steps until there is no significant changes in the weight vectors. The batch Polyceptron algorithm is given below as pseudo code.

Algorithm 1: Batch Polyceptron

Input: Training dataset $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$, K

Output: $\{\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K\}$

begin

Initialize $\tilde{\mathbf{w}}_1^0, \dots, \tilde{\mathbf{w}}_K^0$, $\eta(\cdot)$, criterion γ , $c \leftarrow 0$;

while $\sum_{k=1}^K \eta(c) \|\sum_{\mathbf{x}_n \in S_k^c} t_n \tilde{\mathbf{x}}_n\| < \gamma$ **do**
 $c \leftarrow c + 1$;

for $k \leftarrow 1$ **to** K **do**

$\tilde{\mathbf{w}}_k^c \leftarrow \tilde{\mathbf{w}}_k^{c-1} + \eta(c) \sum_{\mathbf{x}_n \in S_k^{c-1}} t_n \tilde{\mathbf{x}}_n$;

end

end

return $\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K$;

end

B. Online Polyceptron

In the online Polyceptron algorithm, the examples are presented in a sequence. Also at every iteration the algorithm updates the weight vectors based on a single example presented to the algorithm at that iteration. The online algorithm works in the following way. The example \mathbf{x}_c at c^{th} iteration is checked to see whether it is classified correctly using the present set of parameters Θ^c . If the example is correctly classified then all the weight vectors are kept same as earlier. But if \mathbf{x}_c is misclassified and $r = \arg \min_{k \in \{1, \dots, K\}} (\tilde{\mathbf{w}}_k^c)^T \tilde{\mathbf{x}}_c$, then only $\tilde{\mathbf{w}}_r$ is updated in the following way.

$$\tilde{\mathbf{w}}_r^{c+1} = \tilde{\mathbf{w}}_r^c + t_c \tilde{\mathbf{x}}_c \quad (2)$$

Here we have set the step size as '1'. In general, any other appropriate step size can also be chosen. The complete online Polyceptron algorithm is described in Algorithm.2.

In the online Polyceptron algorithm, we see that the contribution to the error from \mathbf{x}_c will be reduced because we have

$$-t_c (\tilde{\mathbf{w}}_r^{c+1})^T \tilde{\mathbf{x}}_c = -t_c (\tilde{\mathbf{w}}_r^c)^T \tilde{\mathbf{x}}_c - t_c^2 \tilde{\mathbf{x}}_c^T \tilde{\mathbf{x}}_c < -t_c (\tilde{\mathbf{w}}_r^c)^T \tilde{\mathbf{x}}_c$$

In the above we have used the fact that $\tilde{\mathbf{x}}_c^T \tilde{\mathbf{x}}_c > 0$. Although the contribution to the error due to \mathbf{x}_c is reduced, but this does not mean that the error contribution of other misclassified examples to the error is also reduced.

As is easy to see that online algorithm is very similar to the standard Perceptron algorithm. The original Perceptron algorithm is known to converge in finite iteration if the training set is linearly separable. However, this does not imply that the Polyceptron algorithm would converge if the data is polyhedrally separable. The reason for this is as follows: when \mathbf{x}_c is misclassified, we are using it to update the weight vector $\tilde{\mathbf{w}}_r$, where r is chosen $r = \arg \min_k t_c \mathbf{x}_c^T \tilde{\mathbf{w}}_k^c$. While this may

Algorithm 2: Online Polyceptron

Input: A sequence of examples $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$,
 K
Output: $\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K$
begin
 Initialize: $\tilde{\mathbf{w}}_1^0, \dots, \tilde{\mathbf{w}}_K^0$, $p = 0$
 while $p < \text{num-passes}$ **do**
 $p \leftarrow p + 1$;
 for $c \leftarrow 1$ **to** N **do**
 Get a new example (\mathbf{x}_c, y_c) ;
 Predict $\hat{y}_c = \text{sign}(h(\mathbf{x}_c))$;
 Let $r = \text{argmin}_{k \in \{1, \dots, K\}} (\tilde{\mathbf{w}}_k^{c-1})^T \tilde{\mathbf{x}}_c$;
 if $\hat{y}_c \neq y_c$ **then**
 $\tilde{\mathbf{w}}_r^c \leftarrow \tilde{\mathbf{w}}_r^{c-1} + y_c \tilde{\mathbf{x}}_c$;
 $\tilde{\mathbf{w}}_k^c \leftarrow \tilde{\mathbf{w}}_k^{c-1}$, $\forall k \neq r$;
 else
 $\tilde{\mathbf{w}}_k^c \leftarrow \tilde{\mathbf{w}}_k^{c-1}$, $k = 1, \dots, K$;
 end
 end
 $\tilde{\mathbf{w}}_k^0 \leftarrow \tilde{\mathbf{w}}_k^N$, $k = 1, \dots, K$;
 end
 return $\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K$;
end

be a good heuristic to decide which hyperplane should take care of \mathbf{x}_c , we have no knowledge of this. This is the same credit assignment problem that we explained earlier. At present we have no proof of convergence of the online Polyceptron algorithm. However, given the empirical results presented in the next section, we feel that this algorithm should have some interesting convergence properties.

IV. EXPERIMENTS

To test the effectiveness of Polyceptron algorithm, we test its performance on several synthetic and real world datasets. We compare our approach with different types of algorithms. We compare our approach with OC1 [10] which is generic top down oblique decision tree algorithm. We compare our approach with a constrained optimization based approach for learning polyhedral classifier discussed in [6]. This approach successively solves linear programs. We call it PC-SLP (Polyhedral Classifier-Successively Linear Program) approach. We also compare our approach with a polyhedral learning algorithm called SPLA1 [9]. Since the objective here is to explicitly learn the hyperplanes that define the polyhedral set, we feel that comparisons with other general PR techniques (e.g., SVM) are not relevant.

Dataset Description

We generate two polyhedrally separable datasets in different dimensions which are described below,

- 1) **Dataset 1: 10-dimensional polyhedral set** 1000 points are sampled uniformly from $[-1 \ 1]^{10}$. A polyhedral set is formed by intersection of following three halfspaces.

- a) $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + 1 \geq 0$
- b) $x_1 - x_2 + x_3 - x_4 + x_5 - x_6 + x_7 - x_8 + x_9 - x_{10} + 1 \geq 0$
- c) $x_1 + x_3 + x_5 + x_7 + x_9 + 0.5 \geq 0$

Points falling inside the polyhedral set are labeled as positive examples and the points falling outside this polyhedral set are labeled as negative examples. The number of positive and negative examples sampled are 493 and 507 respectively.

- 2) **Dataset 2: 20-dimensional polyhedral set** 1000 points are sampled uniformly from $[-1 \ 1]^{20}$. A polyhedral set is formed by intersection of following four halfspaces.

- a) $x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 6x_6 + 7x_7 + 8x_8 + 8x_9 + 8x_{10} + 20x_{11} + 8x_{12} + 7x_{13} + 6x_{14} + 5x_{15} + 4x_{16} + 3x_{17} + 2x_{18} + x_{19} + x_{20} + 20 \geq 0$
- b) $-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 + 6x_6 - 7x_7 + 8x_8 - 9x_9 + 15x_{10} - 11x_{11} + 10x_{12} - 9x_{13} + 8x_{14} - 7x_{15} + 6x_{16} - 5x_{17} + 4x_{18} - 3x_{19} + 2x_{20} + 15 \geq 0$
- c) $x_1 + x_3 + x_5 + x_7 + 2x_8 + 8x_{10} + 2x_{12} + 3x_{13} + 3x_{15} + 3x_{16} + 4x_{18} + 4x_{20} + 8 \geq 0$
- d) $x_1 - x_2 + 2x_5 - 2x_6 + 6x_9 - 3x_{10} + 4x_{13} - 4x_{14} + 5x_{17} - 5x_{18} + 6 \geq 0$

Points falling inside the polyhedral set are labeled as positive examples and the points falling outside this polyhedral set are labeled as negative examples. The number of positive and negative examples sampled are 462 and 538 respectively.

Apart from these two synthetic datasets, we also illustrate the performance of our algorithm on a simple 2-dimensional dataset where the polyhedral set is a square. Here the dataset is obtained by uniformly sampling from $[-2 \ 2] \times [-2 \ 2]$ in \mathbb{R}^2 and the positive examples are those fully inside $[-1.1 \ 1.1] \times [-1.1 \ 1.1]$. This dataset is used only to illustrate how the algorithm learns and for this we show how the polyhedral set being learnt evolves during the iterative optimization procedure.

We also test Polyceptron on two real world datasets downloaded from UCI ML repository [11] which are described in Table I.

Data set	Dimension	# Points
Ionosphere	34	351
Breast-Cancer	10	683

TABLE I
DETAILS OF DATASETS USED FROM UCI ML REPOSITORY

Experimental Setup

We implemented *Polyceptron* in MATLAB. In the batch Polyceptron there are two user defined parameters, namely the step size $\eta(\cdot)$ and the stopping criterion γ . For our experiments, we use a constant step size $\eta = .1$ for all iterations and for all datasets. The γ value used for every dataset is specified in Table.II. In the online Polyceptron, we have one

user defined parameter *num-passes* which is an upper bound on the number of data passes. Different values of parameter *num-passes* are used for different datasets and are mentioned in Table II. We implemented SPLA1 [9] in MATLAB. In SPLA1, the number of hyperplanes are fixed beforehand. We used fixed step size gradient ascent (GA) for the maximization steps in SPLA1. The step size α for SPLA1 is a user defined parameter. For OC1 we have used the downloadable package available from Internet [12]. We implemented PC-SLP approach also in MATLAB. All the user defined parameters for different algorithms are found using ten fold cross validation results. All the simulations were done on a PC (Core2duo, 2.3GHz, 2GB RAM).

Analysis of Experiments

We now discuss performance of Polyceptron in comparison with other approaches on different datasets. The results provided are based on 10 repetitions of 10-fold cross validation. We show average values and standard deviation (computed over 10 repetitions) of accuracy, time taken and the number of hyperplanes learnt. Note that in Polyceptron, we fix the number of hyperplanes beforehand. The results are presented in Table II. We show results of both batch and online Polyceptron. Table II shows results obtained with SPLA1, OC1 and SLP also for comparisons.

We see that batch Polyceptron is always better than online Polyceptron in terms of time and accuracy. In the online Polyceptron at any iteration only one weight vector is changed when the current example is being misclassified. After every iteration the Polyceptron algorithm may not be improving as far as the Polyceptron criterion is concerned. Because of this online Polyceptron takes more time to find appropriate weight vectors to form the polyhedral classifier. On the other hand, batch Polyceptron minimizes the Polyceptron criterion using an alternating minimization scheme. And we observe experimentally that the Polyceptron criterion is monotonically decreased after every iteration using batch Polyceptron.

We see that the batch Polyceptron performs better than SPLA1 in terms of accuracy with a huge margin except on Ionosphere dataset. For Ionosphere dataset also the accuracy of batch Polyceptron is lesser than the accuracy of SPLA1 by a very small amount. Time wise SPLA1 is up to 2.5 times faster than batch Polyceptron except on Ionosphere dataset. For Ionosphere dataset batch Polyceptron runs faster than SPLA1.

The results obtained with OC1 show that a generic decision tree algorithm is not good for learning polyhedral classifier. Polyceptron learns the required polyhedral classifier with lesser number of hyperplanes compared to OC1 which is a generic decision tree algorithm. This happens because we have a model based approach which is specially designed for polyhedral classifiers whereas OC1 is a greedy approach to learn general piecewise linear classifiers. For synthetic datasets, we see that accuracies of both batch and online Polyceptron are greater than that of OC1 with a huge margin. As the dimension increases, the search problem for OC1

explodes combinatorially. As a consequence, performance of OC1 decreases as the dimension is increased which is apparent from the results shown in Table II. Also OC1, which is a general decision tree algorithm gives a tree with a large number of hyperplanes.

For real word datasets, batch Polyceptron outperforms OC1 always. We see that for Breast Cancer dataset and Ionosphere dataset, polyhedral classifiers learnt using batch Polyceptron give very high accuracy. This can be assumed that both these datasets are nearly polyhedrally separable. In general, batch Polyceptron is much faster than OC1.

Compared to PC-SLP [6], Polyceptron approach always performs better in terms of both time and accuracy. As discussed in Section I, SLP which is a nonconvex constrained optimization based approach, has to deal with credit assignment problem combinatorially which degrades its performance both computationally and qualitatively. Polyceptron does not suffer from such problem.

Thus, in summary, Polyceptron, in general, outperforms a generic decision tree method as well as any specialized algorithm for learning polyhedral sets (e.g., PC-SLP).

V. CONCLUSIONS

In this paper, we have proposed a new approach for learning polyhedral classifiers which we call *Polyceptron*. We propose Polyceptron criterion whose minimizer will give us the polyhedral classifier. To minimize Polyceptron criterion, we propose online and batch version of Polyceptron algorithm. Batch Polyceptron minimizes the Polyceptron criterion using an alternating minimization algorithm. Online Polyceptron algorithm works like Perceptron algorithm as it updates the weight vectors only when there is a misclassification. We see that both the algorithm are very simple to understand and implement. We show experimentally that our approach efficiently finds polyhedral classifiers when the data is actually polyhedrally separable. For real world datasets also our approach performs better than any general decision tree method or specialize method for polyhedral sets.

REFERENCES

- [1] N. Megiddo, "On the complexity of polyhedral separability," *Discrete and Computational Geometry*, vol. 3, pp. 325–337, December 1988.
- [2] D. M. J. Tax and R. P. W. Duin, "Support vector domain description," *Pattern Recognition Letters*, vol. 20, pp. 1191–1199, 1999.
- [3] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," in *Knowledge Discovery and Data Mining*, vol. 2, pp. 121–167, 1998.
- [4] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers - a survey," *IEEE Transaction on System, Man and Cybernetics-Part C: Application and Reviews*, vol. 35, pp. 476–487, November 2005.
- [5] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. John Wiley & Sons, second ed.
- [6] A. Astorino and M. Gaudioso, "Polyhedral Separability through Successive LP," *Journal of Optimization Theory and Applications*, vol. 112, pp. 265–293, February 2002.
- [7] C. Orsenigo and C. Vercellis, "Accurately learning from few examples with a polyhedral classifier," *Computational Optimization and Applications*, vol. 38, pp. 235–247, 2007.
- [8] M. Dundar, M. Wolf, S. Lakare, M. Salganicoff, and V. Raykar, "Polyhedral classifier for target detection a case study: Colorectal cancer," in *Proceedings of the twenty fifth International Conference on Machine Learning (ICML)*, (Helsinki, Finland), July 2008.

Data set	Method	Accuracy	Time(sec.)	# hyperplanes
Dataset1	Batch Polyceptron ($\gamma = 50$)	94.34 ± 1.11	0.12 ± 0.001	3
	Online Polyceptron (<i>num-passes</i> =300)	89.08 ± 0.91	1.55 ± 0.01	3
	SPLA1-GA ($\alpha = 0.9$)	90.80 ± 4.93	0.06	3
	OC1	77.53 ± 1.74	6.65 ± 0.87	22.01 ± 5.52
	PC-SLP	71.26 ± 5.46	27.70 ± 10.07	3
Dataset2	Batch Polyceptron ($\gamma = 50$)	94.56 ± 0.90	0.23 ± 0.53	4
	Online Polyceptron (<i>num-passes</i> =400)	94.34 ± 1.96	1.76 ± 0.16	4
	SPLA1-GA ($\alpha = 1.2$)	90.89 ± 2.99	0.09 ± 0.01	4
	OC1	63.64 ± 1.48	10.01 ± 0.67	27.36 ± 6.98
	PC-SLP	56.42 ± 0.79	189.91 ± 19.73	4
Ionosphere	Batch Polyceptron ($\gamma = 20$)	89.06 ± 2.22	0.02 ± 0.002	2
	Online Polyceptron (<i>num-passes</i> =500)	81.15 ± 2.66	0.91 ± 0.03	2
	SPLA1-GA ($\alpha = .3$)	89.65 ± 1.98	0.06 ± 0.004	2
	OC1	86.49 ± 2.08	2.4 ± 0.11	8.99 ± 3.36
	PC-SLP	78.77 ± 3.96	45.31 ± 35.66	2
Breast Cancer	Batch Polyceptron ($\gamma = 20$)	98.49 ± 0.21	0.12 ± 0.03	2
	Online Polyceptron (<i>num-passes</i> =500)	91.93 ± 3.36	1.72 ± 0.01	2
	SPLA1-GA	89.50 ± 5.17	0.05 ± 0.01	2
	OC1	94.89 ± 0.81	1.52 ± 0.13	5.82 ± 0.95
	PC-SLP	83.87 ± 1.42	22.86 ± 1.07	2

TABLE II
COMPARISON RESULTS

- [9] N. Manwani and P. S. Sastry, "Learning polyhedral classifiers using logistic function," *Journal of Machine Learning Research - Proceedings Track*, vol. 13, pp. 17–30, 2010.
- [10] S. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *Journal of Artificial Intelligence Research*, vol. 2, pp. 1–32, 1994.
- [11] A. Asuncion and D. J. Newman, "UCI machine learning repository," 2007. <http://www.ics.uci.edu/~mlearn/MLRepository.html>.
- [12] S. Murthy, S. Kasif, and S. Salzberg, The OC1 decision tree software system, 1993. Software available at <http://www.cs.jhu.edu/~salzberg/announce-oc1.html>.